R4B10 - Cryptographie et Sécurité

SSH

Bruno BEAUFILS

2024/2025

1. Introduction

2. Principe

3. OpenSSH

Secure Shell

Connexion à un ordinateur distant avec accès à un terminal

- accès **distant** : shell, copie de fichier, transfert de ports
- connexion **chiffrée**

Histoire

- conçu pour remplacer rlogin, rsh, rcp, telnet et ftp
- débute en 1995, normalisé en 2006
- 2 versions (ssh1, ssh2)

Détails

① **protocole** de communication

RFCs **4251** ([4252](#), [4253](#), [4254](#), [4255](#))

- ▶ TCP port 22
- ▶ architecture client/serveur

② ensembles de **programmes**

- ▶ implémentation du protocole

Secure Shell

Connexion à un ordinateur distant avec accès à un terminal

- accès **distant** : shell, copie de fichier, transfert de ports
- connexion **chiffrée**

Histoire

- conçu pour remplacer rlogin, rsh, rcp, telnet et ftp
- débute en 1995, normalisé en 2006
- 2 versions (ssh1, ssh2)

Détails

① **protocole** de communication

RFCs **4251** (4252, 4253, 4254, 4255)

- ▶ TCP port 22
- ▶ architecture client/serveur

② ensembles de **programmes**

- ▶ implémentation du protocole

Logiciels

- Architecture client/serveur

- ▶ serveur sur la machine distante
- ▶ clients sur la machine locale

sshd
ssh, sftp, scp

- Plusieurs implémentations

- ▶ **OpenSSH**

- issu de **OpenBSD**
- licence libre
- implémentation complète (serveur **et** client)
- disponible sur beaucoup d'OS

la référence

- ▶ **PuTTY** (émulateur de terminal), **Dropbear**

- ▶ Autres : **serveurs** et **clients**

- Debian

```
apt-get install openssh-server
```

```
apt-get install openssh-client
```

1. Introduction

2. Principe

3. OpenSSH

Chiffrement à clés (rappels)

- Chiffrer = rendre illisible un message
 - ▶ exemple : code de César (ROT13)
 - ▶ vocabulaire : chiffrer $\neq$ crypter
- Chiffrer avec une **clé** = utilisation d'une clé pour chiffrer et déchiffrer
 - ▶ objectif : transmettre un message sans crainte d'une interception
 - ▶ connaître le message chiffré et le codage ne suffit pas à déchiffrer
 - ▶ la clé est
 - juste une information (un nombre par exemple)
 - un paramètre de la fonction de codage
- Chiffrement **symétrique** (à *clé secrète*)
 - ▶ la même clé sert à chiffrer et déchiffrer
 - ▶ les 2 parties doivent connaître la clé avant la communication
- Chiffrement **asymétrique**
 - ▶ 2 clés différentes :
 - une sert à chiffrer
 - une autre sert à déchiffrer
 - ▶ la clé publique peut (doit) être distribuer largement

clé publique
clé privée

Chiffrement à clés (rappels)

- Chiffrer = rendre illisible un message
 - ▶ exemple : code de César (ROT13)
 - ▶ vocabulaire : chiffrer $\neq$ crypter
- Chiffrer avec une **clé** = utilisation d'une clé pour chiffrer et déchiffrer
 - ▶ objectif : transmettre un message sans crainte d'une interception
 - ▶ connaître le message chiffré et le codage ne suffit pas à déchiffrer
 - ▶ la clé est
 - juste une information (un nombre par exemple)
 - un paramètre de la fonction de codage
- Chiffrement **symétrique** (à *clé secrète*)
 - ▶ la même clé sert à chiffrer et déchiffrer
 - ▶ les 2 parties doivent connaître la clé avant la communication
- Chiffrement **asymétrique**
 - ▶ 2 clés différentes :
 - une sert à chiffrer
 - une autre sert à déchiffrer
 - ▶ la clé publique peut (doit) être distribuer largement

clé publique
clé privée

Chiffrement à clés (rappels)

- Chiffrer = rendre illisible un message
 - ▶ exemple : code de César (ROT13)
 - ▶ vocabulaire : chiffrer $\neq$ crypter
- Chiffrer avec une **clé** = utilisation d'une clé pour chiffrer et déchiffrer
 - ▶ objectif : transmettre un message sans crainte d'une interception
 - ▶ connaître le message chiffré et le codage ne suffit pas à déchiffrer
 - ▶ la clé est
 - juste une information (un nombre par exemple)
 - un paramètre de la fonction de codage
- Chiffrement **symétrique** (à *clé secrète*)
 - ▶ la même clé sert à chiffrer et déchiffrer
 - ▶ les 2 parties doivent connaître la clé avant la communication
- Chiffrement **asymétrique**
 - ▶ 2 clés différentes :
 - une sert à chiffrer
 - une autre sert à déchiffrer
 - ▶ la clé publique peut (doit) être distribuer largement

clé publique
clé privée

Chiffrement à clés (rappels)

- Chiffrer = rendre illisible un message
 - ▶ exemple : code de César (ROT13)
 - ▶ vocabulaire : chiffrer $\neq$ crypter
- Chiffrer avec une **clé** = utilisation d'une clé pour chiffrer et déchiffrer
 - ▶ objectif : transmettre un message sans crainte d'une interception
 - ▶ connaître le message chiffré et le codage ne suffit pas à déchiffrer
 - ▶ la clé est
 - juste une information (un nombre par exemple)
 - un paramètre de la fonction de codage
- Chiffrement **symétrique** (à *clé secrète*)
 - ▶ la même clé sert à chiffrer et déchiffrer
 - ▶ les 2 parties doivent connaître la clé avant la communication
- Chiffrement **asymétrique**
 - ▶ 2 clés différentes :
 - une sert à chiffrer
 - une autre sert à déchiffrer
 - ▶ la clé publique peut (doit) être distribuer largement

clé **publique**
clé **privée**

Principe du fonctionnement SSH

- 1 chiffrement du canal (symétrique)
 - ▶ choix de la clé de chiffrement pour la session
- 2 authentification de l'utilisateur
 - ▶ mot de passe
 - ▶ échanges de clés publiques
- 3 travail effectif
 - ▶ shell complet
 - ▶ commande(s)
 - ▶ capture de paquets
 - ▶ lié au shell local (entrée et sortie standards)

Établissement et chiffrement de la connexion

- 1 choix des protocoles à utiliser
 - 1 le serveur envoie les **protocoles** qu'il supporte
 - 2 si le client en trouve un acceptable on continue sinon on s'arrête
- 2 vérification de la clé publique du serveur
 - 1 le serveur envoie sa **clé publique**
 - 2 le client vérifie la clé
 - première connexion : il faut une vérification physique
 - connexions suivantes : comparaison avec la clé enregistrée pour ce serveur
- 3 négociation d'une **clé de session**
 - ▶ une clé pour un chiffrement **symétrique**
 - ▶ utilisée pour tous les échanges pendant la session
 - ▶ négociation par **échange de clés Diffie-Hellman**
 - utilisation clé publique de l'un et clé privée de l'autre

Point faible : confiance dans la clé publique du serveur

- elle est transmise sur le réseau *interception, modification* potentielle
- il faut vérifier son intégrité la première fois
- exemple : comparaison clé reçue et clé physiquement sur le serveur

Établissement et chiffrement de la connexion

- 1 choix des protocoles à utiliser
 - 1 le serveur envoie les **protocoles** qu'il supporte
 - 2 si le client en trouve un acceptable on continue sinon on s'arrête
- 2 vérification de la clé publique du serveur
 - 1 le serveur envoie sa **clé publique**
 - 2 le client vérifie la clé
 - première connexion : il faut une vérification physique
 - connexions suivantes : comparaison avec la clé enregistrée pour ce serveur
- 3 négociation d'une **clé de session**
 - ▶ une clé pour un chiffrement **symétrique**
 - ▶ utilisée pour tous les échanges pendant la session
 - ▶ négociation par **échange de clés Diffie-Hellman**
 - utilisation clé publique de l'un et clé privée de l'autre

Point faible : confiance dans la clé publique du serveur

- elle est transmise sur le réseau *interception, modification* potentielle
- **il faut vérifier son intégrité la première fois**
- exemple : comparaison clé reçue et clé physiquement sur le serveur

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - ▶ le serveur chiffre un message aléatoire avec la clé publique demandée
 - ▶ le serveur envoie le message chiffré au client
 - ▶ le client déchiffre le message

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

● Méthode

- 1 l'utilisateur génère une paire de clé **assymétrique**
- 2 il donne sa clé publique à un utilisateur distant
- 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - a le serveur chiffre un message aléatoire avec la clé publique demandée
 - b le serveur envoie le message chiffré au client

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

Authentification (utilisateur distant)

Pour accéder au serveur il faut que le client prouve qu'il y est **autorisé**

- être capable de prouver être un utilisateur connu du système

Par mot de passe

password authentication

- méthode standard ...
 - ▶ comparaison des versions chiffrées des mots de passe
 - ▶ circulation dans le canal chiffré
 - ▶ envoi du mot de passe entre les 2 machines (client vers serveur)...
- ... mais *protégée* par le chiffrement de la session

Par échange de clés

public key authentication

- Méthode
 - 1 l'utilisateur génère une paire de clé **assymétrique**
 - 2 il donne sa clé publique à un utilisateur distant
 - 3 lors de la connexion vers cet utilisateur le serveur vérifie que le client possède bien la clé privée associée :
 - 1 le serveur chiffre un message aléatoire avec la clé publique demandée
 - 2 le serveur envoie le message chiffré au client
 - 3 le client déchiffre le message
 - 4 le client utilise le message et la clé de session pour calculer un hash cryptographique
 - 5 le client envoie le hash au serveur
 - 6 le serveur calcule le même hash
 - 7 le serveur compare le hash calculé et celui reçu

1. Introduction

2. Principe

3. OpenSSH

Serveur : clés de serveur, configuration

Localisation

- clé privée du serveur `/etc/ssh/ssh_host_ecdsa_key`
- clé public du serveur `/etc/ssh/ssh_host_ecdsa_key.pub`

Manipulation

- afficher une empreinte (raccourci, *fingerprint*) de la clé publique

```
ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub
```

Configuration

- daemon `sshd_config(5)`
 - ▶ `/etc/ssh/sshd_config` : configuration du serveur
 - ▶ exemples de directives (valeurs par défaut sous Debian)

```
PermitRootLogin prohibit-password
PubkeyAuthentication yes
PermitEmptyPasswords no
AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2
```

- utilisateur distant
 - ▶ `$HOME/.ssh/authorized_keys` : clés publiques autorisées à se connecter
 - format définit dans `sshd(8)`
 - permet de limiter les autorisations à des commandes par exemple

Client : génération de clés

- `ssh-keygen(1)`

- ▶ comme l'authentification par clés est recommandée
- ▶ il faut pouvoir fabriquer des paires de clés

```
ssh-keygen -t TYPE -f FICHIER -C COMMENTAIRE
```

- TYPE

- ▶ `dsa`, `ecdsa`, `ecdsa-sk`, `ed25519`, `ed25519-sk`, `rsa`

- FICHIER

- ▶ chemin de base des fichiers
 - FICHIER = clé privée
 - FICHIER.pub = clé publique
- ▶ par défaut `$HOME/.ssh/id_TYPE`
- ▶ identifie la paire de clés

- clé privée stockée dans FICHIER

- ▶ si les droits sont trop ouverts ssh ne fait pas confiance dans la clé
- ▶ droits maximum 0600

- *passphrase* demandée lors du processus

- ▶ utilisée pour chiffrer le contenu du fichier (clé de chiffrement)
- ▶ nécessaire à ssh à **chaque** utilisation de la clé privée
- ▶ si elle est vide la clé n'est pas chiffrée

Client : conservation des clés en mémoire

- `ssh-agent(1)`
 - ▶ parce que saisir la passphrase à chaque utilisation est contraignant...
 - ▶ ...conserver en **mémoire** les clés privées
 - ▶ démarré en début de session de travail
 - ▶ arrêté en fin de session de travail
 - ▶ communique avec `ssh` via
 - `$SSH_AUTH_SOCK` : une socket UNIX
 - `$SSH_AGENT_PID` : le PID du process agent
- `ssh-add(1)` : manipuler les clés gérés par `ssh-agent`

```
# lister les clés publiques des paire de clés en mémoire  
ssh-add -L
```

```
# lister les empreintes des clés publiques des paires en mémoire  
ssh-add -l
```

```
# ajouter une paire de clés en mémoire  
ssh-add FICHIER
```

```
# retirer une paire de clés de la mémoire  
ssh-add -d FICHIER
```

```
# retirer toutes les paires de clés de la mémoire  
ssh-add -D
```

Client : copie de la clé publique sur le serveur

- `ssh-copy-id(1)`
 - ▶ pour être autorisé à se connecter
 - la clé publique utilisée doit être présente sur la machine distante
 - dans `$HOME/.ssh/authorized_keys` (pour l'utilisateur distant)
 - ▶ `ssh-copy-id` fait la copie **proprement**
 - on peut s'en passer mais ce serait dommage

```
ssh-copy-id -i FICHER USER@MACHINE
```

- copie la clé publique sur MACHINE
- ajoute la clé publique dans le `$HOME/.ssh/authorized_keys` de USER
- lors de la première copie l'authentification se fait par mot de passe

Outils : utilisation

ssh(1)

`ssh USER@MACHINE COMMANDE`

- connexion sur MACHINE en tant que USER
- si pas de COMMANDE on démarre un shell sur MACHINE
 - ▶ sinon on exécute COMMANDE dans un processus sur MACHINE
- connexion transparente des fichiers du processus distant au shell local
 - ▶ `stdin`, `stdout`, `stderr`
- accès complet (tubes et redirections utilisables)

scp(1)

- Copie avec origine **ou** destination distante
 - ▶ quasiment la même syntaxe que la commande `cp`
- Syntaxe pour identifier le *distant*

utilisateur @ machine : chemin

sftp(1)

- Transfert interactif de fichiers à distance
 - ▶ fonctionnement similaire à `ftp(1)` mais dans un canal chiffré
 - ▶ permet de manipuler les fichiers distants
- Syntaxe pour identifier le *distant* identique à `ssh(1)`

Outils : transfert

Transfert de connexions par port (tunnel)

- rediriger les connexions d'un port local sur un port distant
 - ▶ options : `-L PORT_LOCAL:MACHINE_DEPUIS_DISTANTE:PORT_DISTANT`
 - ▶ les données reçues sur la machine locale sur `PORT_LOCAL` ...
 - ▶ ... sont envoyées sur `MACHINE_DEPUIS_DISTANTE:PORT_DISTANT` à partir de `MACHINE`
- rediriger les connexions d'un port distant sur un port local
 - ▶ options : `-R PORT_DISTANT:MACHINE_DEPUIS_LOCAL:PORT_LOCAL`
 - ▶ les données reçues sur `MACHINE` sur `PORT_DISTANT` ...
 - ▶ ... sont envoyées sur `MACHINE_DEPUIS_LOCAL:PORT_LOCAL` à partir de la machine locale

Transfert des clés en mémoire

- rediriger les demandes de clés par un ssh distant vers l'agent local
- `ForwardAgent`

Outils : configuration du comportement

- comportement défini par les options de la ligne de commande
- comportement peut-être défini comme standard (par défaut)
 - ▶ via des directives dans `$HOME/.ssh/config`
 - ▶ chaque option a une directive équivalente
 - utilisable en ligne de commande via `-o`
 - ▶ voir `ssh(1)` et `ssh_config(5)`
- comportements identifiables par un *alias*
 - ▶ directive `Host`
 - ▶ tous les directives suivantes sont lié à l'alias
- exemple
 - ▶ `$HOME/.ssh/config`

```
ForwardAgent yes
Host QuiEstSurTP
    Hostname tp.iutinfo.fr
    User bruno.beaufils
    RemoteCommand who | tr -s " " | cut -d " " -f 1 | sort | uniq
```

- ▶ appel

```
$ ssh QuiEstSurTP
augustin.beeuwsaert.etu
bruno.beaufils
```

Résumé des outils

- commandes

- ▶ **ssh(1)** : commande principale
- ▶ **sshd(8)** : daemon SSH
- ▶ **ssh-keygen(1)** : gestion des clés asymétriques
- ▶ **ssh-copy-id(1)** : copie d'une clé publique sur un compte distant
- ▶ **ssh-agent(1)** : agent d'authentification (conserve les clés en mémoire)
- ▶ **ssh-add(1)** : manipulation des clés gérées par ssh

- configuration

- ▶ dossier de configuration
 - machine : /etc/ssh
 - utilisateur : \$HOME/.ssh
- ▶ serveur sshd_config(5)
 - /etc/ssh/sshd_config : configuration du serveur
 - \$HOME/.ssh/authorized_keys : clés autorisées (cf sshd(8))
- ▶ client
 - \$HOME/.ssh/known_hosts : clés publiques des serveurs de confiance (connus)
 - \$HOME/.ssh/id_rsa : clé RSA privée par défaut
 - \$HOME/.ssh/id_rsa.pub : clé RSA publique par défaut
- ▶ configuration par défaut du comportement des outils ssh_config(5)
 - machine : /etc/ssh/ssh_config
 - utilisateur : \$HOME/.ssh/config

Résumé des outils

- commandes

- ▶ `ssh(1)` : commande principale
- ▶ `sshd(8)` : daemon SSH
- ▶ `ssh-keygen(1)` : gestion des clés asymétriques
- ▶ `ssh-copy-id(1)` : copie d'une clé publique sur un compte distant
- ▶ `ssh-agent(1)` : agent d'authentification (conserve les clés en mémoire)
- ▶ `ssh-add(1)` : manipulation des clés gérées par ssh

- configuration

- ▶ dossier de configuration
 - machine : `/etc/ssh`
 - utilisateur : `$HOME/.ssh`
- ▶ serveur `sshd_config(5)`
 - `/etc/ssh/sshd_config` : configuration du serveur
 - `$HOME/.ssh/authorized_keys` : clés autorisées (cf `sshd(8)`)
- ▶ client
 - `$HOME/.ssh/known_hosts` : clés publiques des serveurs de confiance (connus)
 - `$HOME/.ssh/id_rsa` : clé RSA privée par défaut
 - `$HOME/.ssh/id_rsa.pub` : clé RSA publique par défaut
- ▶ configuration par défaut du comportement des outils `ssh_config(5)`
 - machine : `/etc/ssh/ssh_config`
 - utilisateur : `$HOME/.ssh/config`

Références

- pages du manuel
- <https://www.openssh.com/>
- WikiBook OpenSSH